Rapport TER L2

Application en C++ d'aide à l'organisation des tâches

Meryll Essig - Tamara Rocacher - Ludovic Fanus - Céleste Bonkeke

26 avril 2015

Remerciements

Nous souhaitons remercier notre encadrant de projet, Abdelhak-Djamel Seriai, pour l'aide apportée tout au long du projet, et pour les réunions régulières qui nous ont permis de garder la finalité du projet en tête. D'autre part, nous voulons également remercier Christian Retore pour son implication générale dans cette UE, autant que pour la mise à disposition, en temps et en heures, des diverses informations qui nous ont permit de mener ce projet à bien.

Table des matières

1 Introduction **2**

2 Domaine d'étude **3**
2.1 Développement logiciel 3
2.2 Administration de base de données 3

3 Sujet de travail **4**
3.1 Problématique 4
3.2 Besoins 4

4 Conception **7**
4.1 Fonctionnalités 7
4.2 Interface 9
4.3 Base de données 9

5 Réalisation **11**
5.1 Interface 11
5.2 Base de données 12

6 Conclusion **13**

A Extraits de code **14**

B Captures d'écran **18**

C Manuel d'utilisation **21**

Chapitre 1

Introduction

Notre groupe est constitué de 4 étudiants de deuxième année d'informatique, puis a été divisé en deux sous groupes, lors de la réalisation. Cela nous a permis d'effectuer un travail en parallèle. Nous avons donc choisit un coordinateur de projet pour améliorer les communications au sein du groupe et avec notre encadrant. Nous avons choisit un sujet de projet se situant dans le domaine du développement logiciel, pour nous permettre d'améliorer nos capacités de programmation par la pratique. D'autre part, nous sommes amenés à gérer une base de données, ce qui nous a permis de découvrir ce domaine auquel nous avons été initiés en début d'année. Le sujet propose donc la création d'une application d'aide à l'organisation des tâches, que nous avons choisit d'implémenter avec Qt, pour le graphisme. Le système n'étant pas multi-utilisateurs (BDD locale) et les données à stocker étant assez limitées, nous avons choisit d'utiliser SQLite pour notre base de données. En vue de rendre compte de manière fidèle et analytique du travail réalisé au cours du semestre, nous souhaitons présenter au préalable le domaine d'étude de notre projet, puis la problématique à laquelle nous avons souhaité répondre par ce travail, avant de détailler chaque étape de la conception à la réalisation de notre application.

Chapitre 2

Domaine d'étude

En tant qu'étudiants en deuxième année de licence en informatique, nous souhaitions travailler sur un projet qui nous permette de mettre en pratique nos acquis, afin de les consolider. Ainsi, notre projet d'application d'aide à l'organisation des tâches se situe dans le domaine du développement logiciel, qui nous amène à parfaire notre maitrise du langage C++, couplé à l'administration de base de données, suscitant la mise en pratique des cours du semestre 3.

2.1 Développement logiciel

Ce domaine [1] regroupe les différentes étapes necessaires à la mise en place et au suivit du bon fonctionnement d'un logiciel, repondant à un ou des besoins utilisateurs. On y retrouve donc :

- l'analyse des besoins et des scénarios d'interactions, permettant d'atteindre au mieux le but visé,
- la conception, servant à determiner les solutions techniques les mieux adaptées à satisfaire le cahier des charges, qui amène à produire des documents tels que le diagrammes de classes UML (Unified Modeling Language) et le modèle de données,
- la construction, qui est la partie la plus importante , en temps, du devellopement d'un logiciel. Cette phase regroupe la rédaction du code source, ainsi que la mise en place d'un "mode d'emploi" (ou document d'aide) et des fichiers de configuration necessaires à la portabilité de l'exécutable final.
- la recette, conscistant en une série de test et de contrôles précédant la mise sur le marché du logiciel. Pour cela, une présérie (version béta) est mise à disposition d'un petits nombre d'utilisateurs qui évalue la cohérence entre le logiciel et les attentes (cahier Des Charges).

Pour notre projet, une version béta sera testée par notre entourage et éventuellement par des professeurs de l'université, afin de repérer d'éventuels bogues ou de nous faire part de la facilité ou non de l'utilisation.

2.2 Administration de base de données

L'administration de base de données [2] permet de garantir le bon fonctionnement des serveurs de bases de données d'après sa conception de bases, et la gestion (controle et protection) de leurs utilisations. Cela regroupe la modélisation, assurant l'intégrité des données et du fonctionnement ; la définition de l'espace de stockage, délimité par les données à stocker ; la création de la base de données, selon de standards choisit (nommage, procédures...)

Chapitre 3

Sujet de travail

3.1 Problématique

Le sujet choisit par notre groupe est l'implémentation d'un logiciel d'aide à l'organisation des tâches. Ainsi, nous avons définit la problématique de notre projet comme étant de rendre un service à un utilisateur qui souhaite pouvoir organiser son temps de manière simple et efficace, avoir accès rapidement à son emploi du temps et être éventuellement averti avant le début d'une tâche prévue.

En effet, une application d'aide à l'organisation des tâches tient un rôle d'agenda, permettant de noter chaque tâche (réunions, dates importantes, activités et loisirs, voire tâches ménagères...) à mesure que l'utilisateur en prend connaissance. De plus, l'application joue un rôle de bloc note, car étant virtuelle (pas de format papier limité en écriture), l'utilisateur peut conserver des remarques personnelles ou des informations importantes concernant une tâche (ex : liste de course, materiel à apporter...) sans contrainte et de façon organisée, plus facilement. Enfin, l'avantage d'avoir un agenda numérique est de pouvoir être prévenu dans un délai déterminé avant le début d'une tâche, ce qui permet à l'utilisateur d'être assuré de ne pas oublier une tâche importante de son planning.

Cependant de nombreux logiciels ou applications, tel que Google Calendar, proposent déjà ces services, enrichis de diverses fonctionnalités facilitant la saisie des tâches ou encore offrant un graphisme perfectionné et épuré. Toutefois, nous avons constaté que pour toutes les applications existantes que nous avons consulté, l'utilisateur doit déjà avoir une idée globale de son planning et des heures des différentes tâches qu'il veut compulser. Ainsi, la problématique à été élargie à la possibilité de proposer à l'utilisateur de générer automatiquement son plannig selon des critères de son choix , tels que la priorité d'une tâche, les jours et heures qui lui conviennent le mieux, etc.

3.2 Besoins

Pour implémenter de telles fonctionnalités, il était necessaire de déterminer les différents besoins de l'utilisateur auxquels nous souhaitions répondre. Nous avons donc déterminé les suivants :

Saisie des tâches

Le premier besoin de l'utilisateur est de pouvoir conserver des informations concernant une tâche à effectuer : l'objet de la tâche (intitulé), la date prévue de début, la date de fin, le lieu...

Tâches récurrentes

L'utilisateur peut avoir des tâches "types", qui reviennent régulièrement, tel qu'un cours de musique hebdomadaire ou un entrainement sportif quotidient. Ainsi, un des besoins principaux est de pouvoir déterminer la récurrence (quotidienne, hebdomadaire, mensuelle) d'une tâche afin de ne pas avoir à saisir de nouveau les mêmes informations à chaque occurrence.

Rappel

Autre besoin primordial, la necessité de se rappeler des tâches à effectuer, au moment opportun. Le rappel automatique correspond donc à un besoin de l'utilisateur, lorsqu'il s'agit d'un agenda numérique, lui permettant de s'assurer sans s'en soucier, de ne pas oublier une tâche importante.

Catégories

L'utilisateur peut avoir plusieurs tâches concernant un même type d'activités (ex : études, constitué de plusieurs cours, révisions, travaux a rendre...) et regroupant des informations communes. Il aura donc besoin de determiner plus facilement une catégorie de tâches, conservant les données qui lui sont attribuées et permettant, dans un même temps, une meilleure organisation de ses tâches.

Ensemble de tâches

La conservation d'une tâche necessite d'avoir des dates de début et de fin (ou une durée) précises, cependant certaines tâches peuvent elles-mêmes amener l'utilisateur à exécuter différentes sous-tâche sur une longue période de temps. Ainsi, l'utilisateur aura besoin de pouvoir préciser et consulter les différentes sous-tâches réalisées et restantes ainsi que d'être éventuellement avertit avant le début de certaines d'entre elles. La notion d'ensemble de tâches permet alors de créer plusieurs tâches et sous tâches reliées entre elles dans un ordre précis et respectant une échéance commune.

Agenda

L'utilisateur recquiert un affichage de ses tâches à venir dans l'ordre chronologique et selon différentes vues, facilitant la lecture et la recherche de la tâche la plus proche ou d'une tâche précise. Il aura donc besoins de différentes vues de son agenda, lui permettant de le visualiser dans son ensemble (ex : lors de la prise d'un rendez-vous), ou de vérifier plus rapidement les tâches immédiatement à venir(début de journée ou de semaine).

Plannification Automatique

Selon l'appronfondissement de notre problématique, nous avons déterminé qu'un besoin de l'utilisateur manquait. En effet, l'utilisateur peut vouloir effectuer certaines tâches sans avoir d'impératifs concernant l'heure ou la date précise de ces tâches. Cependant, il aura toujours une estimation des échéances, de la durée ou encore des heures approximatives où il préferera effectuer ce type de tâche. Ainsi, le besoin est de pouvoir communiquer ces informations et d'en tirer une plannification des tâches concernées sans y réflechir (contraintes avec les autres tâches fixées, les heures ou jour de repos, le manque de temps...).

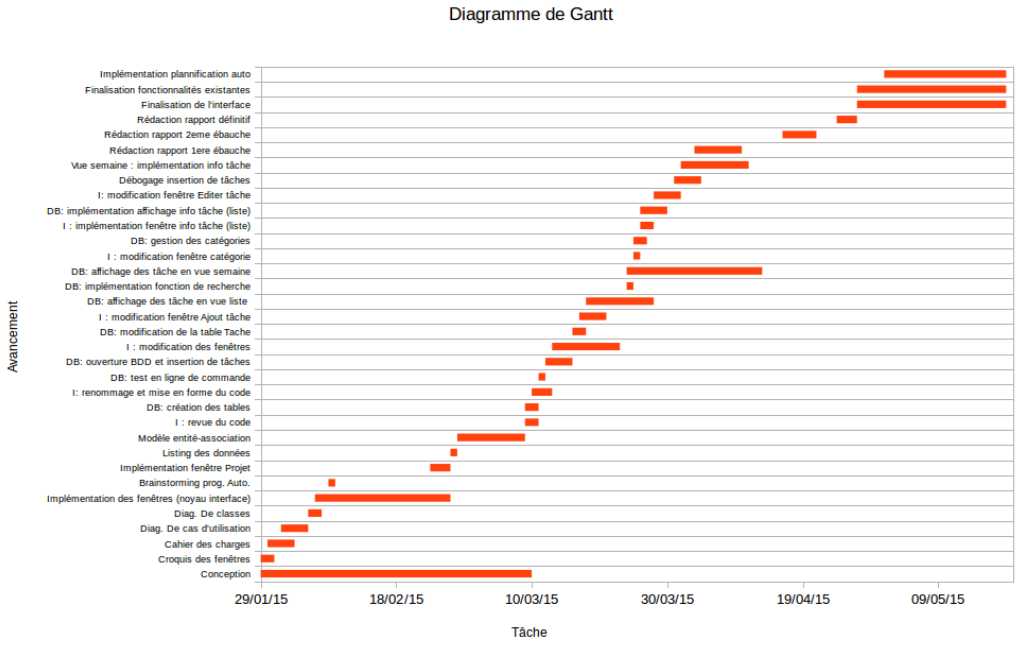

FIGURE 3.1 – Diagramme de Gantt

Recherche

Un besoin des plus basiques est de pouvoir consulter et vérifier ou modifier une tâche précise. Pour cela il faut donc la trouver. Ainsi en découle le besoin d'une recherche selon différents critères (catégorie, intitulé, jour...), permettant un accès rapide, autrement dit de ne pas voir a parcourir tout le planning dans le cas ou l'utilisateur ne se rappel plus, par exemple, de la date précise.

Chapitre 4

Conception

Dès le début du projet, nous nous sommes décidés d'un avis commun à utiliser Qt, qui permet une bonne gestion des divers outils et/ou fonctionnalités qu'il nous faudrait utiliser par la suite, notamment la gestion de la mémoire (allocation dynamique simplifiée) ou encore des bases de données ; en plus de nous donner un accès d'utilisation relativement simple à une bibliothèque graphique très compléte.

4.1 Fonctionnalités

Grâce à la réalisation d'un diagramme de cas d'utilisations, les fonctionnalités correspondant aux besoins préalablement définis ont pu être précisées et approfondies, mettant en évidences les différentes actions de l'utilisateur lors de l'interaction avec l'application. Cette phase à notemment permit de mettre en lumière les différents points de vue des membres du groupe et de considérer toutes les possibilités pour en tirer les moyens les mieux adaptés pour répondre aux besoins définis.

Au fil des réunions, nous avons soulevé, avec l'aide de notre encadrant, de nouveaux besoins jusqu'alors non pris en compte, et qui nous ont permis d'enrichir notre conception de l'application. Ainsi, nous avons prévu l'espace de programmation, qui repond au besoin d'une plannification automatique, ou encore, par la suite, la fonction de recherche de tâche.

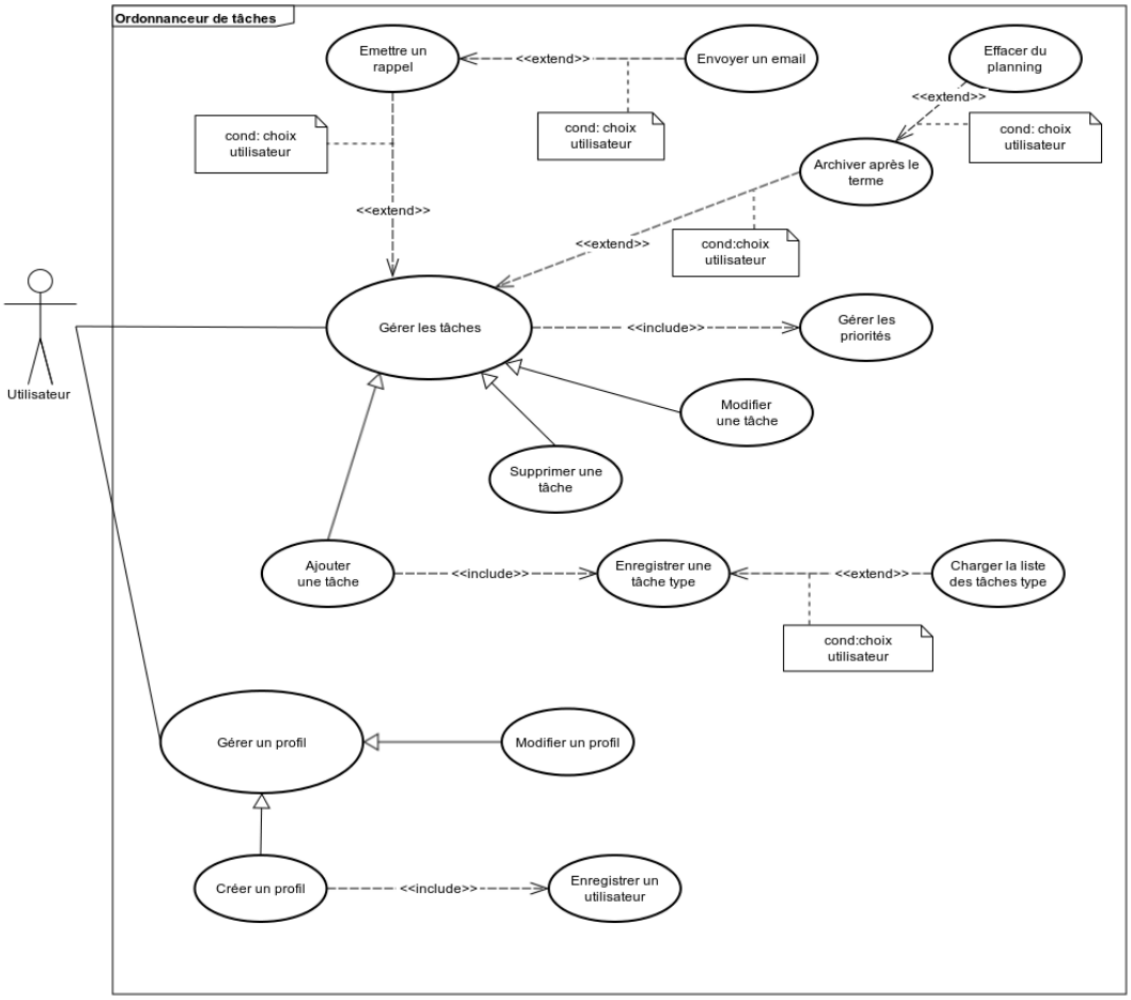

FIGURE 4.1 – Diagramme de cas d'utilisation

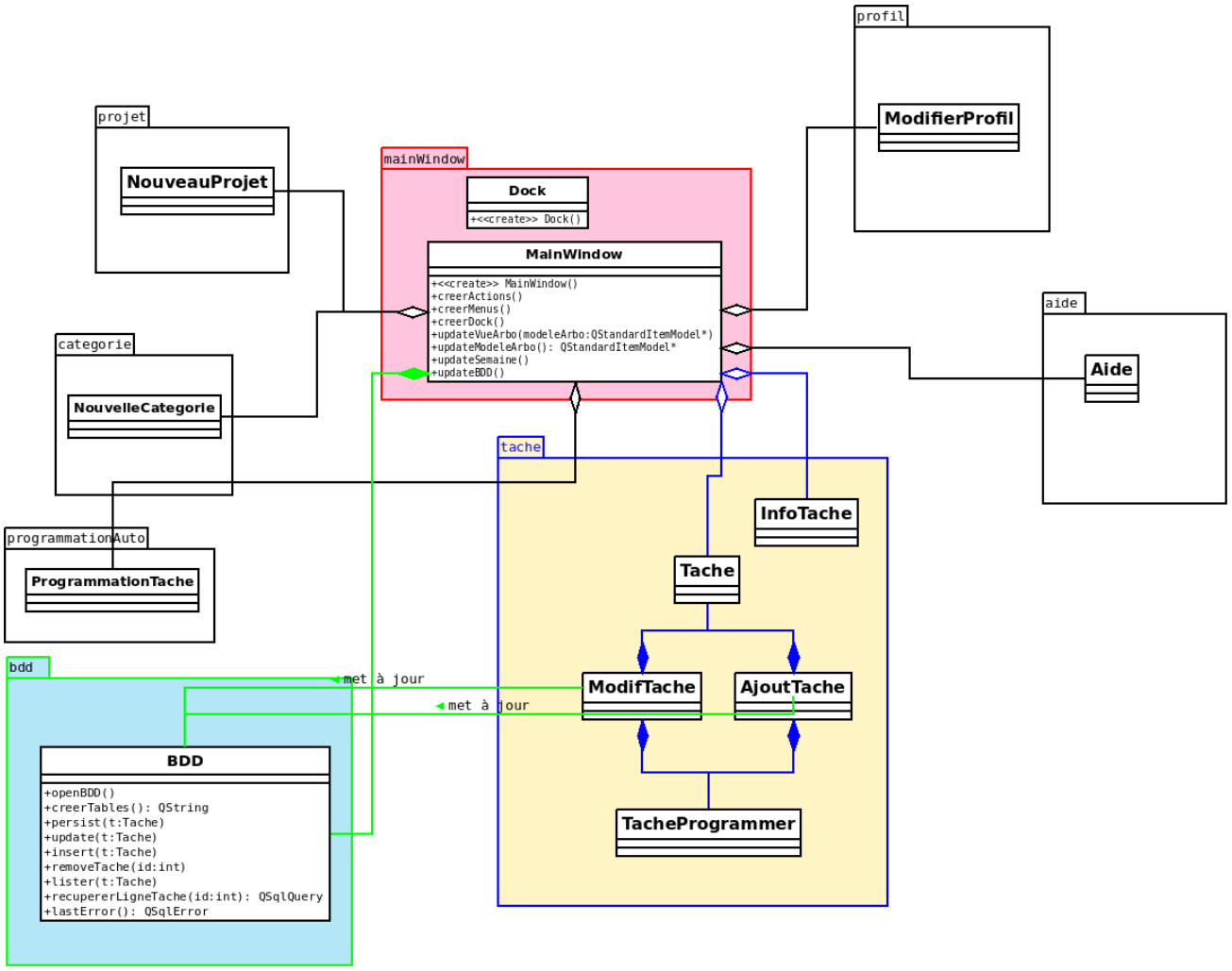

FIGURE 4.2 – Diagramme de classes

4.2 Interface

Après la définition globale des fonctionnalités, nous avons constitué le diagramme de classes, par le biais duquel nous avons définit l'essentiel de l'interface graphique. En effet, pour pouvoir déterminer correctement l'ensemble des classes qui nous seraient nécessaires, nous avons du établir une liste exhaustive de l'ensemble des fenêtres utilent à l'utilisateur, que ce soit pour la saisie ou pour l'affichage. Par la suite, nous avons déterminé les connexions et interactions des différentes fenêtres entre elles, permettant de prévoir les méthodes essentielles à leur bon fonctionnement. De plus, plusieurs schémas ont été représentés afin de décider d'une présentation de l'interface la plus adaptée et considérant les différentes idées du groupe. A partir de là, nous sommes passés l'implémentation de l'interface, avant de reprendre la conception de la partie Base De Données.

4.3 Base de données

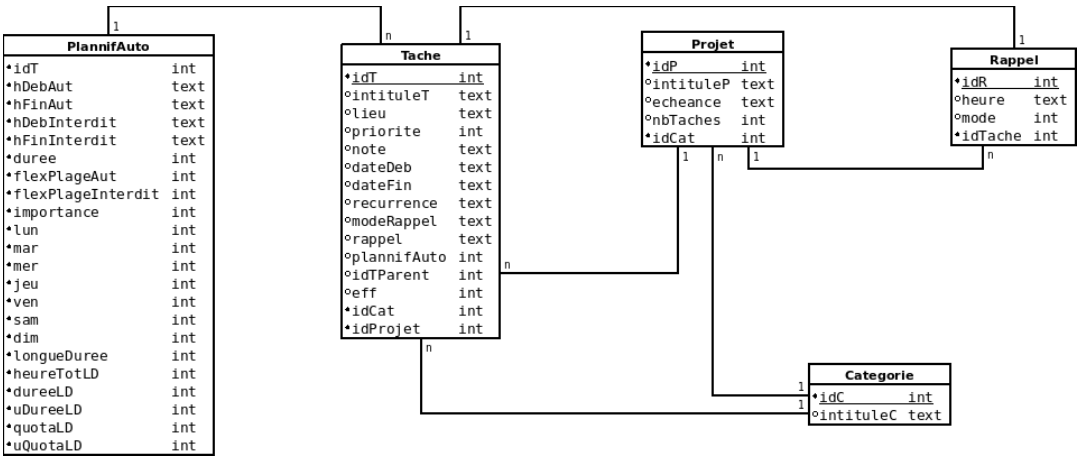

FIGURE 4.3 – Modèle entié-association

Pour commencer, un modèle entité-association à été établit à partir des différentes classes fixées lors de la conception. Une fois ce modèle validé, le groupe a été divisé en deux sous-groupe, l'un attitré à l'interface et l'autre consacré à la base de données. Ainsi, le groupe concerné à dressé une liste exhaustive de toutes les données pouvant être renseignées par l'utilisateur et devant donc être stockées. Le choix du Système de Gestion de Base de Données s'est porté sur SQLite pour sa simplicité et ses fonctionnalités suffisantes pour les besoins de notre application. A partir de là, et sur l'appuie du modèle entité-association, les différentes tables ont été constituées, avec leurs données correspondantes. Cinq tables ont ainsi été determinées pour la base de données, qui sera stockée sur l'ordinateur même de l'utilisateur, l'application étant prévue pour être utilisée par une seule et même personne sur chaque ordinateur (non multi-utilisateur).

Chapitre 5

Réalisation

La réalisation s'est déroulée en deux temps, le premier commençant avec l'implémentation basique et essentielle de l'interface, repartie sur l'ensemble des membres du groupe en fonction des différentes fenêtres (+/- 2 fenêtres par membre). La seconde phase de réalisation à permit un travail parallèle des deux sous-groupes, l'un adaptant et perfectionnant l'interface et l'autre s'occupant principalement de la gestion de la base de données.

5.1 Interface

Notre interface à été la première phase de l'implémentation, nous permettant d'obtenir rapidement une référence visuelle sur laquelle travailler. Chaque membre du groupe à implémenté les classes correspondant aux différentes fenêtres lui ayant été attribuées. Par la suite cette interface a beaucoup évolué, avec l'avancement des fonctionnalités et la connexion avec la base de données. Nous avons choisit d'implémenter chaque fenêtre distincte par une classe C++, et de garder le même schéma de construction pour chacune (Annexe A.1 et A.2). Cette partie n'a pas posé de problème particulier au niveau du code mais à pris du temps pour prendre en compte toutes les informations nécessaires aux différentes fonctionnalités et adapter l'ergonomie de l'application, afin de garder une simplicité d'utilisation. Ainsi nous avons été amenés à créer de nouvelles fenêtres ou séparer une fenêtre existante en sous-fenêtres, pour éviter une surcharge visuelle et conserver la cohérence du regroupement des informations. Notamment, la fenêtre de saisie des tâches, étant la fenêtre de saisie principale pour l'utilisateur, à été totalement revue au fur et à mesure de la phase de conception de la base de données. Cette fenêtre recoupe de nombreuses imformations, ouvrant éventuellement sur d'autres fenêtres (ex : création de catégorie personnalisée), et pouvant concerner, en plus de la tâche elle-même, la programmation automatique. Ainsi, cette fenêtre est rapidement devenue trop grande et trop remplie, ne facilitant pas la compréhension et la simplicité d'utilisation. En concertation avec le fonctionnement de la base de données, il à donc été décidé de la subiviser, en regroupant dans une sous-fenêtre (créant une nouvelle classe) toutes les informations liées à la programmation automatique de la tâche. Cette solution à été possible car l'utilisateur n'est pas obligé de choisir la programmation automatique et n'a donc pas l'utilité de voir tous ces champs a chaque création de tâche.

5.2 Base de données

Après la validation du modèle entité-association, la création des tables associées à été faite rapidement, puis nous avons pu tester cette base de données en lignes de commande. Pour cela, nous avons exécuté plusieurs requêtes d'insertion et de récupération de données, notamment à l'aide d'un site de génération automatique de données[3]. Ces tests ont permit de vérifier le bon comportement de la base de données, c'est à dire si chaque table recevait les bonnes valeurs et si chaque requête renvoyait les résultats attendus. Lors de la connexion de cette base de données au reste de l'application, nous avons du créer de nouvelles classe, afin de séparer la vue et le modèle, pour l'affichage des tâches dans les différentes vues. Ainsi, des méthodes de classe C++ ont pu être créées indépendamment, permettant de rafraichir le modèle : les requêtes à la base de données sont renvoyées et si de nouvelles tâches ont été ajoutées elles sont récupérées. Cela permet de gérer l'affichage coté vue sans se soucier du nombre ou de la nature des informations à afficher. Cependant l'algorithme d'affichage à nécessité beaucoup de travail pour obtenir le résultat arborescent, notamment sur la constitution des requêtes. (Annexe A.3). De plus, pour ne pas surcharger la vue liste, nous avons réalisé une fenêtre d'information sur les tâches, nécessitant de récupérer l'identifiant de la tâche sélectionnée au double-clique. Pour cela, nous avons créé une colonne cachée dans le modèle de la liste arborescente afin d'y insérer cet identitifiant unique. Ainsi, l'utilisateur ne voit pas cette information qui lui est inutile, mais nous pouvons récupérer dans la vue, l'information existante dans cette colonne cachée, permettant alors de lancer la requête de récupération des informations utiles de la tâche sélectionnée uniquement.

De la même manière, il nous a fallut simplifier la vue semaine, et la vue mois, qui ne pouvaient présenter dans une seule case plusieurs tâches, au risque d'être illisibles et surchargées. Ainsi, pour la vue semaine, nous avons conçut un algorithme (Annexe A.4), qui a causé des difficultés au niveau des requêtes et de la gestion des dates et heures avec SQLite, mais qui a finalement permis d'afficher l'intitulé de la première tâche de chaque heure puis le nombre de tâche supplémentaire au cours de la même heure, le cas échéant.

A ce stade de la réalisation, nous avions apporté beaucoup de changement, tout en observant les résultats obtenus à partir des bases de données déjà existantes sur nos différents ordinateurs, et contenants plusieurs tâches. Ainsi, en tentant de créer une nouvelle tâche, nous nous sommes confrontés à problème d'insertion dans la base de données. Après plusieurs tests de chaque ancienne version de l'application conservées sur git, nous nous sommes rendus compte que cette erreur existait depuis longtemps et que nous n'avions pas pensé à tester chaque fonctionnalité déjà existante régulièrement. Pour déboguer, nous avons récupéré le code de la dernière version fonctionnelle, puis ajouté peu à peu les différentes modifications réalisées, en testant la compilation à chaque fois. Cela à mis en évidence l'erreur au niveau de la requête d'insertion d'une nouvelle tâche dans la table associée. En effet, lors de l'ajout des valeurs aux données (Annexe A.4) , il y avait plus de " ?" dans le texte de la requête que de "addBindValue()" associés, causant un conflit. Nous avons donc tiré avantage de ce problème pour la suite de notre projet, en faisant plusieurs vérifications à chaque grosse modification du code, ce qui nous permet de corriger de petites erreurs à temps, notamment quand il s'agit de coquille ou d'erreur d'inattention, difficiles à revoir après un certain temps et pouvant causer des bogues importants.

Chapitre 6

Conclusion

Ce projet nous à permis de perfectionner notre maitrise du langage C++, mais aussi de découvrir la bibliothèque Qt et l'IDE QtCreator, très adapté à son utilisation et très fonctionnel. De plus, la réalisation de cette application nous à permis de mettre en pratique les notions apprises au cours du semestre 3 concernant les bases de données et les requêtes. D'autre part, cela nous a appris des méthodes de travail en groupe et la necessité de répartir les différentes tâches à réaliser, mais aussi l'importance de la communication afin de pouvoir bénéficier d'une bonne entente et surtout d'un avancement réel du travail global au fur et à mesure que chacun progresse dans son travail personnel. Pour finir, nous avons aussi appris à utiliser le logiciel de gestion de version décentralisé git [4], et a nous servir des différentes fonctionnalités proposées, comme la création de différentes branches de travail, pour chacun des sous groupes.

Annexe A

Extraits de code

Listing A.1 – Exemple d'interface simple sur Qt : .h

```
#ifndef NOUVELLECATEGORIE\_H
#define NOUVELLECATEGORIE\_H

#include <QWidget>
#include <QFormLayout>
#include <QLineEdit>
#include <QLabel>
#include <QCheckBox>
#include <QPushButton>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QComboBox>
#include <QMessageBox>

class NouvelleCategorie : public QWidget
{

    Q_OBJECT

public :
    NouvelleCategorie();

private:
    QLineEdit *ligneIntitule;
    QPushButton *boutonValider;
    QPushButton *boutonAnnuler;

    QHBoxLayout *layoutBoutons;
    QVBoxLayout *layoutPrincipal;

private slots:
    void ajouterCategorie();
};


#endif // NOUVELLECATEGORIE\_H
```

Listing A.2 – Exemple d'interface simple sur Qt : .cpp

```
#include <QApplication>
#include <QWidget>
#include <QSqlQuery>
#include <QSqlDatabase>
#include "../bdd/BDD.h"

#include "NouvelleCategorie.h"

NouvelleCategorie::NouvelleCategorie() : QWidget()
{
    setWindowTitle("Nouvelle categorie");

    ligneIntitule = new QLineEdit;
    boutonValider = new QPushButton("Valider");
    boutonAnnuler = new QPushButton("Annuler");

    layoutBoutons = new QHBoxLayout;
    layoutPrincipal = new QVBoxLayout;

    layoutBoutons->addWidget(boutonValider);
    layoutBoutons->addWidget(boutonAnnuler);

    layoutPrincipal->addWidget(new QLabel("Intitule de la categorie"));
    layoutPrincipal->addWidget(ligneIntitule);
    layoutPrincipal->addLayout(layoutBoutons);

    setLayout(layoutPrincipal);

    connect(boutonAnnuler, SIGNAL(clicked()), this, SLOT(close()));
    connect(boutonValider, SIGNAL(clicked()), this, SLOT(ajouterCategorie()));
}
```

Listing A.3 – Algorithme de mise à jour du modèle de données pour la vue arborescente

```
1  QStandardItemModel* MainWindow::updateModeleArbo()
2  {
3      modeleArbo = new QStandardItemModel(0,4);
4      QList<QString> listeTitres;
5      listeTitres << "Tache" << "Identifiant" << "Date debut" << "Date Fin";
6      QStringList *titres =new QStringList(listeTitres);
7
8      modeleArbo->setHorizontalHeaderLabels(*titres);
9
10     QStandardItem *ligneEnCoursDeTraitement = new QStandardItem();
11     QStandardItem *sousLigne = new QStandardItem;
12
13     QSqlQuery qParent;
14     QSqlQuery qEnfant;
15     int i=0, count=0, count2 =0;
16
17     qParent.exec("select idT, intituleT, dateDeb, dateFin from Tache where idTParent=0
           and eff=0 order by datetime(dateDeb)");
18
19     while(qParent.next() && count<11){
20         ligneEnCoursDeTraitement=new QStandardItem(qParent.value(1).toString());
21         modeleArbo->appendRow(ligneEnCoursDeTraitement);
22         count++;
23         i=qParent.value(0).toInt();
24
25         qEnfant.prepare("select idT, intituleT, dateDeb, dateFin from Tache where
               idTParent=? and strftime('%Y-%m-%d',dateDeb)>=date((select dateDeb From Tache
                where idT=?)) order by datetime(dateDeb)");
26         qEnfant.bindValue(0,i);
27         qEnfant.bindValue(1,i);
28         qEnfant.exec();
29
30         //Traitement colonnes
31         modeleArbo->setItem(count-1, 1, new QStandardItem(qParent.value(0).toString()));
32         modeleArbo->setItem(count-1, 2, new QStandardItem(qParent.value(2).toString()));
33         modeleArbo->setItem(count-1, 3, new QStandardItem(qParent.value(3).toString()));
34
35         while(qEnfant.next() && count2<5){
36             sousLigne = new QStandardItem(qEnfant.value(1).toString());
37             ligneEnCoursDeTraitement->appendRow(sousLigne);
38             count2++;
39
40             //Traitements colonnes
41             ligneEnCoursDeTraitement->setChild(count2-1,1, new QStandardItem(qEnfant.
                   value(0).toString()));
42             ligneEnCoursDeTraitement->setChild(count2-1,2, new QStandardItem(qEnfant.
                   value(2).toString()));
43             ligneEnCoursDeTraitement->setChild(count2-1,3, new QStandardItem(qEnfant.
                   value(3).toString()));
44         }
45         count2=0;
46     }
47     return modeleArbo;
48 }
```

Listing A.4 – Algorithme de mise à jour du modèle de données de la vue semaine

```
1  //Requete de selection des taches et algo d'affichage de la premiere tache + nb de taches
       suivantes a la meme heure
2      QSqlQuery q;
3      QDate qdateJourTraite = dateCourante.addDays(-numJourSem+1);//on commence au lundi
           puis on ajoutera 1 jour en fin de boucle, jusqu'a 7
4
5      for(int k=0;k<7;k++)//boucle permettant de repeter la requete pour chaque jour de la
           semaine
6      {
7          QString qstringJourTraite = qdateJourTraite.toString("yyyy-MM-dd");//pour
               comparaison where de la requete
8
9          q.prepare("select strftime('%H',dateDeb),count(*) from Tache where date(dateDeb)
               =? group by strftime('%H',dateDeb) ;");
10         q.addBindValue(qstringJourTraite);
11         q.exec();
12
13         int heureEnCours = 25; // Permet de passer les q.next tant que l'heure ne change
               pas, a 25 pour que la premiere fois, il ne corresponde a aucune heure.
14
15         while(q.next()){
16             int nbTacheSup = q.value(1).toInt()-1;
17             QString stringNbTacheSup = "";
18
19
20             if (q.value(1).toInt()>1){// count(idT)>1 cad plus qu'une tache a l'heure en
                   cours
21                 stringNbTacheSup +="    (+"+QString::number(nbTacheSup)+")";
22             }
23
24             heureEnCours = q.value(0).toInt();
25             QSqlQuery qInt;
26             qInt.prepare("select idT,intituleT from Tache where strftime('%H',dateDeb)=?
                   order by dateDeb;");
27             qInt.addBindValue(q.value(0));
28             qInt.exec();
29
30             if(qInt.next()){
31                 modeleSemaine->setItem(heureEnCours,qdateJourTraite.dayOfWeek()-1,new
                       QStandardItem(qInt.value(1).toString()+stringNbTacheSup));
32             }
33         }
34
35         //fin de traitement des taches du jour k, jour suivant ou fin de boucle
36         qdateJourTraite=qdateJourTraite.addDays(1);//passage au jour suivant (+1 jour)
37         qstringJourTraite = qdateJourTraite.toString();//maj de la chaine contenant le
               jour pour comparaison du where dans la requete
38     }
39     vueSemaine->setModel(modeleSemaine);
```

Listing A.5 – Requete d'insertion de tache

```
1  QSqlQuery q;
2      q.prepare("INSERT INTO Tache(intituleT,lieu,priorite,note,dateDeb,dateFin,recurrence,
           modeRappel, rappel, plannifAuto, idCat, idTParent,idProjet,eff) VALUES
           (?,?,?,?,?,?,?,?,?,?,?,?,?,?);");
3      q.addBindValue(t.getIntitule());
4      q.addBindValue(t.getLieu());
5      q.addBindValue(t.getPriorite());
6      q.addBindValue(t.getNote());
7      q.addBindValue(t.getJour().toString("yyyy-MM-dd hh:mm:ss"));
8      q.addBindValue(t.getEcheance().toString("yyyy-MM-dd hh:mm:ss"));
9      q.addBindValue(t.getRecurrence());
10     q.addBindValue("");
11     q.addBindValue(t.getRappel());
12     q.addBindValue(0); //Plannif Auto
13     q.addBindValue(t.getIdC()); //idCat
14     q.addBindValue(0); //idTParent
15     q.addBindValue(t.getIdP());
16     q.addBindValue(0); //Eff
17     q.exec();
```

Annexe B

Captures d'écran

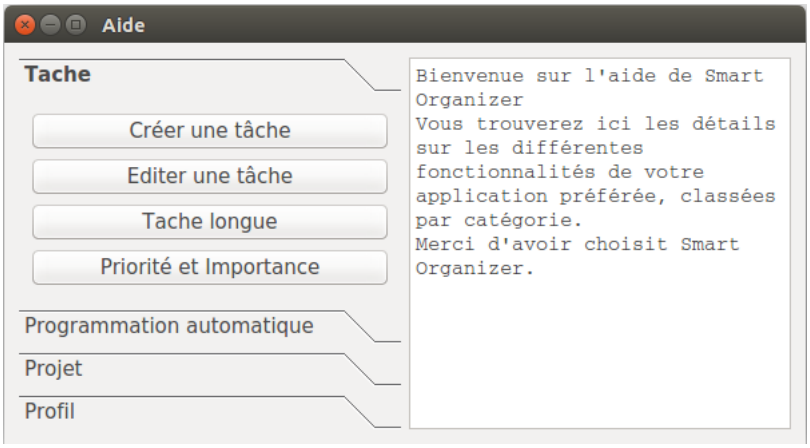

FIGURE B.1 – Fenêtre d'aide

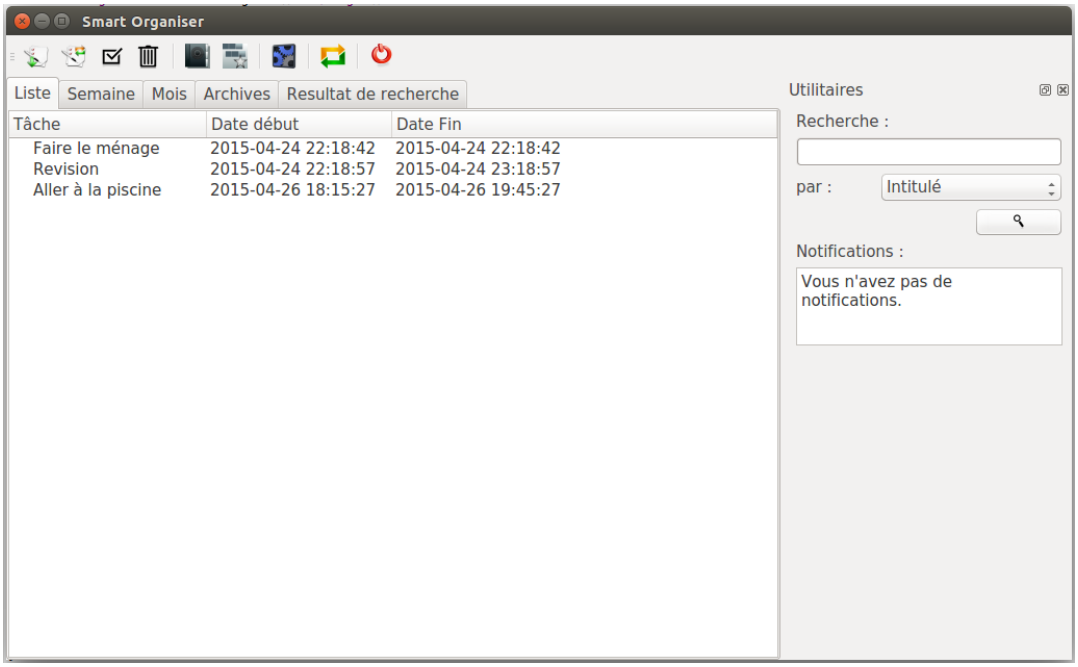

FIGURE B.2 – Fenêtre Principale

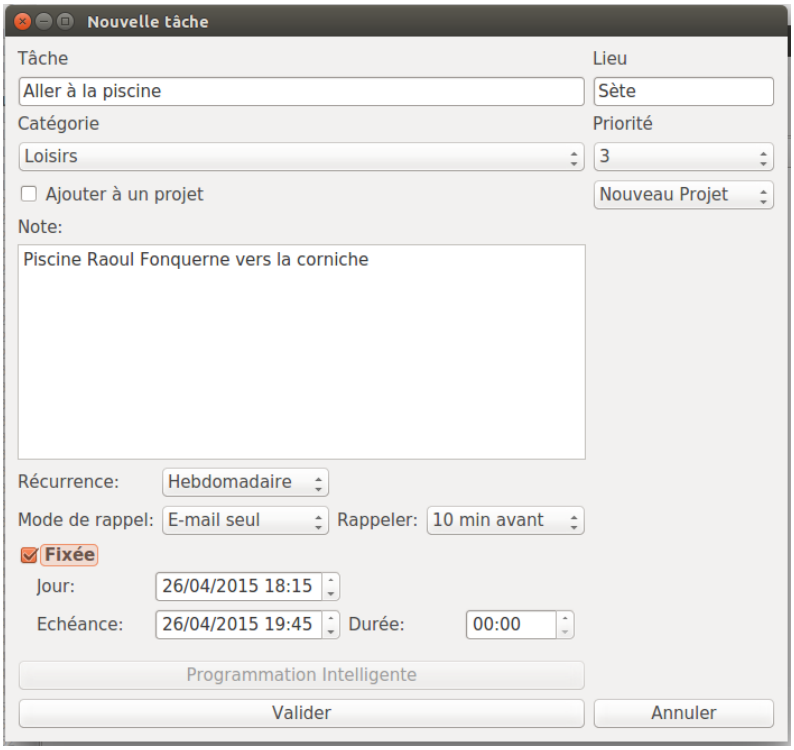

FIGURE B.3 – Fenêtre d'ajout tâche

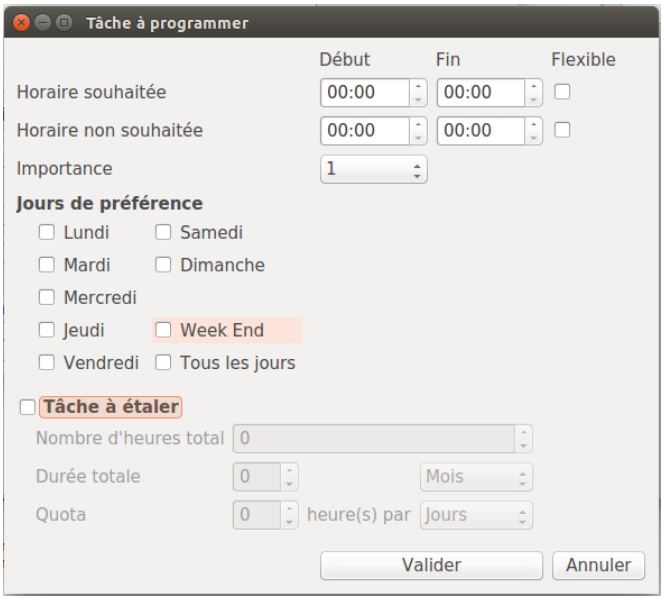

FIGURE B.4 – Paramètres de programmation des tâches

FIGURE B.5 – Fenêtre de programmation des tâches

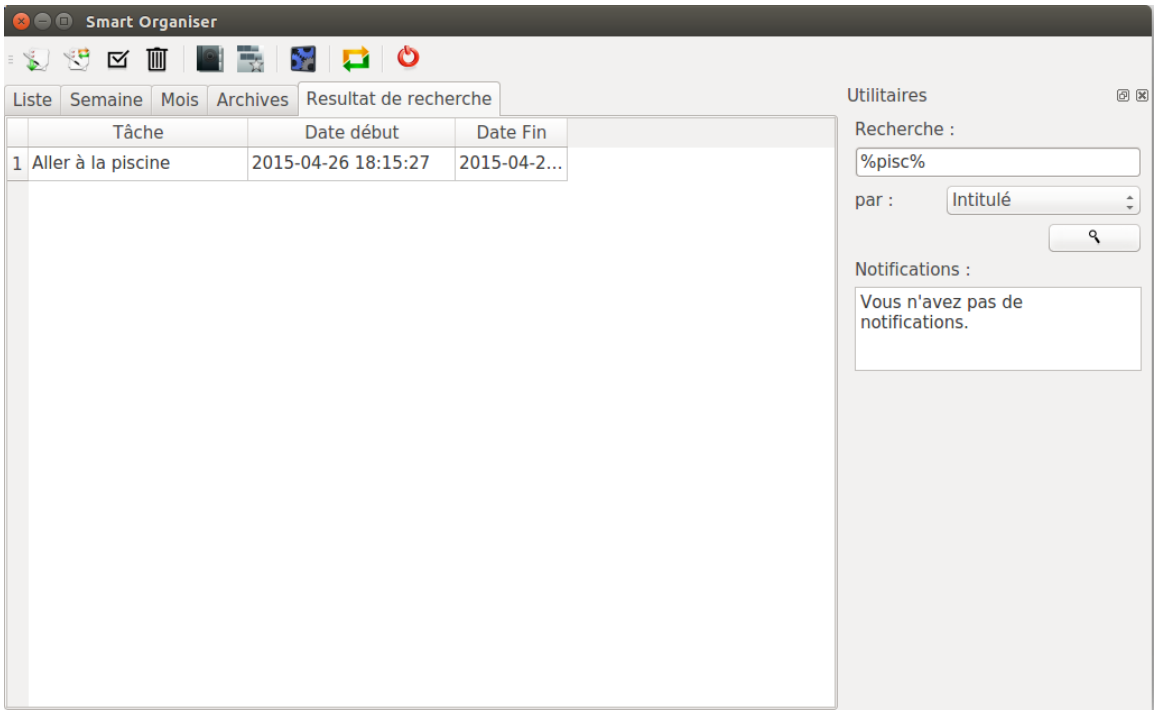

FIGURE B.6 – Rechercher des tâches

Annexe C

Manuel d'utilisation

L'application est simple d'utilisation. Elle ne requiert aucune installation pour ne pas être intrusive.
Ouvrez l'application. Lors de la première utilisation, une fenêtre d'aide s'ouvrira, contenant des informations pour l'utilisation du programme (Voir Annexe B.1).

La fenêtre principale (Annexe B.2) est le centre de l'application. Toutes les informations importantes y sont présentes.

La page à onglet permet d'afficher les tâches selon différentes vues (Liste arborescente, par semaine, par mois) et de voir le résultat d'une recherche de tâche.

Pour créer une nouvelle tâche, cliquez sur *Nouvelle Tâche* dans la barre des menus ou sur l'icône associée dans la barre d'outils. La procédure est similaire pour la création de projet ou la modification d'une tâche.

Dans la fenêtre de nouvelle tâche, vous pourrez choisir vos préférences, et par exemple choisir de laisser l'application gérer intelligement les tâches (Bouton *"Programmation intelligente"* dans l'annexe B.3)

Pour supprimer une tâche, cliquez sur le bouton *Supprimer Tâche* dans la barre de menus ou sur l'icone associée dans la barre d'outils. Il vous faut au préalable avoir selectionné une tâche dans la vue liste.

Pour revoir des informations complètes sur une tâche, il vous suffit de double-cliquer sur la tâche dans la vue liste. Une fenêtre dédiée vous affichera les informations que vous recherchez.
Vous pouvez également créer un projet, qui contiendra de multiples tâches, ajoutables grace au bouton '+' et supprimables grâce au bouton '-'.

Bibliographie

[1] Développement de logiciel. fr.wikipedia.org/wiki/Développement_de_logiciel.

[2] Administrateur de bases de données. fr.wikipedia.org/wiki/Administrateur_de_bases_de_donn%C3%A9es.

[3] Générateur de données. www.generatedata.com.

[4] Notre repository sur github. github.com/tam34/task-manager.